BAB 2
TINJAUAN PUSTAKA

2.1 Profil Perusahaan

PT Bina San Prima Bogor sebagai salah satu anak cabang dari PT Bina San Prima yang bergerak dalam bidang distributor. Berikut sejarah singkat mengenai perusahaan.

2.1.1 Sejarah Perusahaan

PT Bina San Prima didirikan pada tahun 1994, tujuan awal pendirian adalah untuk mengatasi kendala modal keuangan distributor lokal dan kesetaraan distribusi,yang menghambat pertumbuhan bisnis PT Sanbe Farma Pada tahun 2000 (pasca krisis moneter tahun 1998), PT Bina SanPrima membuka 10 cabang sekaligus dalam 1 (satu) tahun.

Sejak tahun 2003, PT Bina San Prima telah menjadi perusahaan Penanaman Modal Asing.Legenda Capital Pte.Ltd, yang berkantor pusat di Singapura, merupakan pemegang saham utama dan pemegang saham sekunder adalah Mr Jahja Santoso.

PT. Sanbe Farma didirikan pada tahun 1975 di Bandunzg oleh Jahja Santoso bersaudara.Nama Sanbe merupakan singkatan dari Santoso bersaudara (Jahja Santoso adalah seorang apoteker lulusan ITB). Kegiatan utama yaitu memproduksi dan juga menjual obat-obatan.

PT. Sanbe pada mulanya memproduksi obat-obat etikal, tahun 1985 Sanbe memproduksi juga obat-obatan untuk hewan.Tahun 1992, Sanbe mulai memasuki pasar obat bebas (OTC) dengan salah satu merek andalannya yaitu Sanaflu.PT. Sanbe Farma menduduki peringkat tertinggi diantara perusahaan farmasi di Indonesia berdasarkan laporan IMS dengan jumlah karyawan lebih dari 1500 orang.

2.1.2 Visi dan Misi

Adapun Visi dan Misi perusahaan adalah sebagai berikut:

a. Visi

Visi adalah suatu pandangan jauh tentang perusahaan, tujuan - tujuan perusahaan dan apa yang harus dilakukan untuk mencapai tujuan tersebut pada masa yang akan datang. Berikut visi pada perusahaan yaitu:

Memimpin dalam pelayanan yang sangat baik melalui Manajemen Solid dan Profesional untuk memberikan nilai signifikan untuk kepentingan nasabah dan kepala.

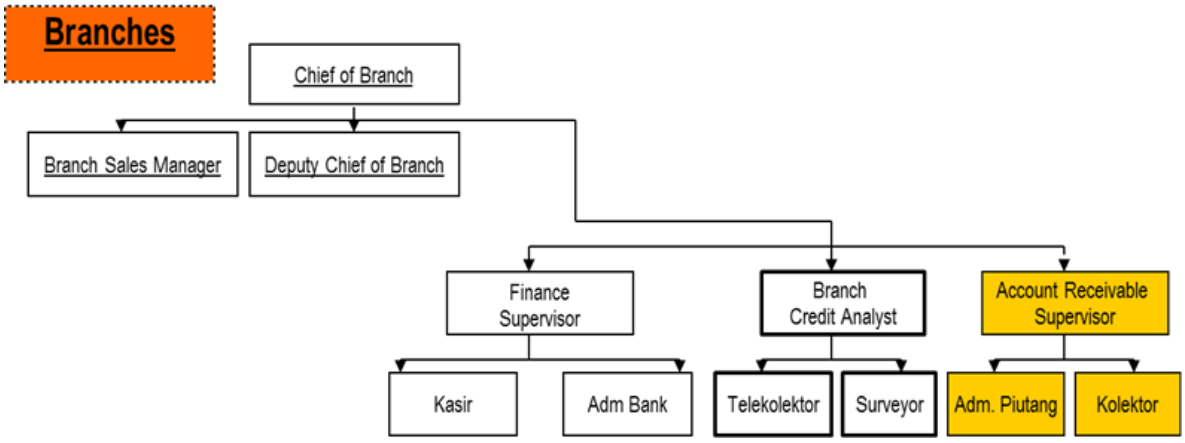
b. Misi

Misi adalah pernyataan tentang apa yang harus dikerjakan oleh lembaga dalam usahanya mewujudkan Visi. Berikut misi pada perusahaan yaitu:

1. Untuk melaksanakan dan untuk mencapai standar distribusi praktek yang baik.
2. Untuk melanjutkan perbaikan pada jaringan luas dan daerah.
3. Untuk mencapai ketersediaan produk yang luas kepala sekolah kami dan pertumbuhan faktur untuk memperkuat daya saing dan kepemimpinan pasar.

2.1.3 Stuktur Organisasi

Organisasi adalah suatu kesatuan yang teratur beberapa kelompok orang yang saling bekerjasama untuk mencapai tujuan bersama. Dalam organisasi diperlukan adanya bentuk dan susunan struktur organisasi untuk menunjukkan kerangka dan susunan perwujudan pola tetap hubungan–hubungan diantara fungsi–fungsi, bagian–bagian atau posisi–posisi, maupun orang–orang yang menunjukkan kedudukan, tugas dan wewenang serta tanggung jawab yang berbeda dalam organisasi. Adapun unsur–unsur organisasi diantaranya kelompok orang dan tujuan bersama. Stuktur organisasi yang ada dapat dilihat pada gambar 2.1.



Gambar 2.1 Stuktur Organisasi

2.1.4 Logo

Logo merupakan suatu bentuk gambar atau sekedar sketsa dengan arti tertentu, dan mewakili suatu arti dari perusahaan, daerah, perkumpulan, produk, negara, dan hal-hal lainnya yang dianggap membutuhkan hal yang singkat dan mudah diingat sebagai pengganti dari nama sebenarnya. Berikut logo pada perusahaan seperti yang tercantum pada gambar 2.2.



Gambar 2.2 Logo Perusahaan

Arti dari gambar 2.2 logo perusahaan adalah sebagai berikut :

Warna biru	: Antusiasme, Spiritualitas, Percaya Diri, Stabilitas
Gradasi	: Cepat (FAST)
Elips	: Continuous Improvement
Anak Panah	: Koordinasi dan sinergi antara BSP dengan Prinsipal
BSP di dalam elips	: Kebersamaan dan kerjasaman
Tulisan BSP	: Kokoh, menjadi the best distributor

2.2 Landasan Teori

Landasan teori berisi teori-teori pendukung yang digunakan untuk membangun sistem aplikasi penarikan *multithread*. Landasan teori yang akan digunakan yaitu *Thread, Multithread,* algoritma penjadwalan, jaringan, client-server, konsep perancangan berorientasi objek, jaringan semantik, MySQL, Borland Delphi & Microsoft Excel.

2.2.1 *Thread*

Suatu proses didefinisikan oleh *resource* yang digunakan dan lokasi yang akan dieksekusi. Namun ada kalanya proses-proses akan menggunakan *resource* secara bersama-sama. Suatu unit dasar dari CPU utilization yang berisi program counter, kumpulan register, dan ruang stack disebut dengan *thread* atau *lighweigt process* (LWP). *Thread* akan bekerja sama dengan *thread* yang lainnya dalam hal penggunaan bagian kode,bagian data, dan *resource* system operasi, seperti open file dan sinyal secara kolektif yang sering disebut dengan *task* [9].

Thread terdiri atas dua bentuk yaitu:

a. *UserThread*

   Thread yang pengaturannya dilakukan oleh pustaka *thread* pada tingkatan pengguna. Karena pustaka yang menyediakan fasilitas untuk pembuatan dan penjadwalan *thread, thread* pengguna cepat dibuat dan dikendalikan.

b. *KernelThread*

Thread yang didukung langsung oleh *kernel*. Pembuatan penjadwalan dan manajemen *thread* dilakukan oleh *kernel* pada *kernel* space. Karena dilakukan oleh sistem operasi, proses pembuatannya akan lebih lambat jika dibandingkan dengan *threaduser* [9].

2.2.1.1 *Multithreading*

Sebuah *processor*, *multithreading* secara umum berlangsung dengan cara *time-division multiplexing* (*multitask ing*) : sebuah *processor* berpindah dari satu *thread* ke *thread* yang lain. Hal tersebut terjadi secara cepat sehingga *user* akan beranggapan bahwa suatu *thread* atau *task* berjalan secara bersamaan. Pada suatu *multiprocessor* atau multi-*core* sistem, suatu *thread* akan dijalankan bersamaan, dengan setiap *processor* atau *core* menjalankan sebuah *thread* atau *task* . Dukungan *thread* dalam bahasa pemrograman bervariasi : tidak semua memiliki bahasa pemrograman mendukung lebih dari satu eksekusi dalam sebuah program untuk dijalankan secara bersamaan [9].

Perbedaan dasar dari *thread* dan processes adalah :

1. Processes pada umumnya independen, sementara *thread* merupakan subset dari sebuah process
2. Processes memiliki alamat berbeda, sedangkan *thread* saling membagi alamat memori.
3. *Contextswitching* antara *thread* pada process yang sama cenderung lebih cepat dibandingkan *contextswitching* antara berbeapa process.

Suatu aplikasi dapat melakukan pekerjaan yang hampir sama oleh banyak *client*. Akan dapat terjadi suatu situasi seperti berikut :

1. Sebuah web *server* menerima banyak *request* dari *client* untuk menampilkan suatu web page.
2. Jika suatu *server* menggunakan teknologi tradisional (*single – thread*) maka web *server* hanya akan dapat melakukan satu pelayanan *request* dalam waktu yang bersamaan.

Ada beberapa pendekatan yang dapat diambil untuk mengatasi masalah tersebut. Dengan pendekatan *multi – threading* kita dapat melakukan hal – hal sebagai berikut :

Solusi pertama :

1. Jika suatu *server* menerima sebuah *request*, *server* tersebut akan membuat suatu proses yang terpisah yang nantinya akan melakukan *request* tersebut.
2. Proses ini sangat memakan waktu dan menggunakan *resource* yang cukup besar.

Solusi kedua :

Jika suatu *request* dibuat, suatu *server* akan menciptakan suatu *thread* lain untuk melayani *request* tersebut [9].

2.2.1.2 Keuntungan *Multithreading*

Multithreading merupakan model *programming* dan pengeksekusian yang populer yang memungkinkan untuk banyak *thread* pada suatu process. *Thread* tersebut merupakan bagian dari process yang sama tapi dapat dieksekusikan secara terpisah. Penggunaan teknologi ini dapat digunakan untuk membuat sebuah process dieksekusikan pada banyak *processor*.

Keuntungan tersebut menjadikan pengeksekusian suatu operasi berjalan lebih cepat pada komputer yang memiliki banyak CPU, CPU yang memiliki banyak inti (*core*), atau operasi antar banyak mesin (*cluster of machines*). Hal ini dikarenakan *thread* dalam program dijalankan secara paralel.

Keuntungan lain yang diperoleh dengan teknologi *multithreading*, walaupun hanya dengan suatu komputer yang memiliki sebuah CPU, yaitu kemampuan untuk sebuah aplikasi untuk tetap *respons*ive terhadap *input*. Pada sebuah program yang memiliki sebuah *thread*, jika sedang mengeksekusikan sebuah *task* yang lama, seluruh aplikasi tersebut akan tampak berhenti (*freeze*). Dengan memindahkan *task* tersebut ke *workerthread* yang berjalan secara paralel dengan eksekusi *thread* utamanya, sangat memungkinkan untuk aplikasi tersebut untuk tetap *respons*ive terhadap *input* dari *user* selama mengerjakan *task* tersebut di latar (*background*).

Sebuah sistem operasi penjadwalan *thread* ada dua cara:

1. *Preemptivemultithreading* dianggap pendekatan yang lebih unggul, hal ini dikarenakan sistem operasi dapat menentukan kapan *contextswitch* harus dilakukan.
2. *Cooperativemultithreading*, merupakan suatu pendekatan dimana *thread* tersebut yang menentukan kapan *thread* tersebut akan diberhentikan.

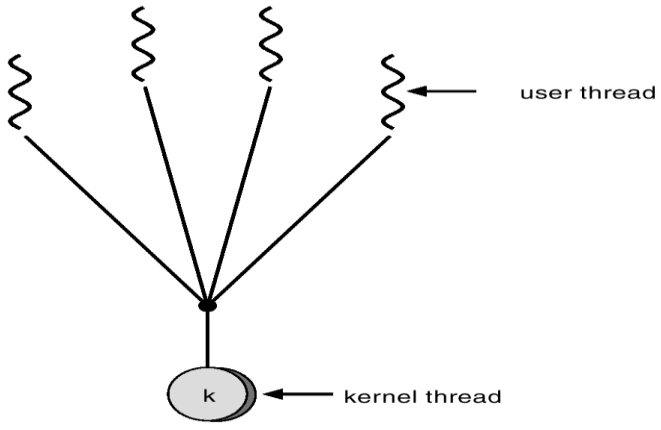
Perangkat keras komputer tradisional tidak memiliki banyak dukungan untuk fasilitas *multithreading* hal ini dikarenakan perpindahan antar satu *thread* dengan *thread* yang lain dianggap lebih cepat dibandingkan keseluruhan proses *contextswitch*. Pada akhir 1990, suatu ide untuk menjalankan instruksi menggunakan banyak *thread* secara bersamaan dikenal dengan nama *simultaneousmultithreading*. Fitur ini diperkenalkan oleh perusahaan Intel dengan nama teknologi *hyperthreading*.

2.2.1.3 Model *Multithreading*

Multithreading secara umum terbagi menjadi tiga model yaitu:

1. *Many to One*

Berikut gambar model multithreading *many to one* seperti yang tercantum pada gambar 2.3.



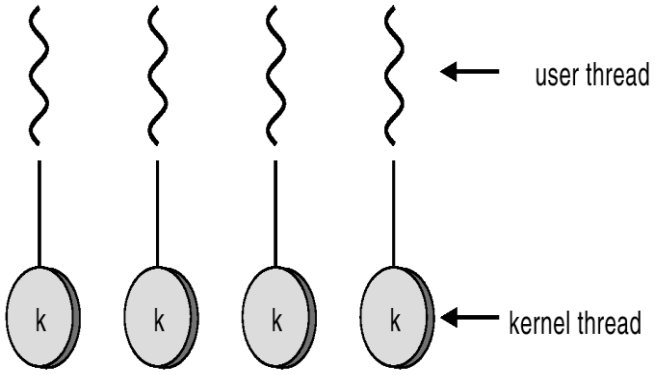
Gambar 2.3 *Many to One* [9]

1) Model ini memetakan beberapa *thread* tingkatan pengguna ke sebuah *thread* tingkatan *kernel*.

2) Pengaturan *thread* dilakukan dalam ruang pengguna sehingga efisien.
3) Hanya satu *thread* pengguna yang dapat mengakses *threadkernel* pada satu saat. Jadi multiple *thread* tidak dapat berjalan secara pararel pada multiprrosesor.
4) Contoh : Solaris Green *Thread*s dan GNU *PortabelThread*s.

2. *One by one*

Berikut model multithreading *one by one* seperti yang tercantum pada gambar 2.4.

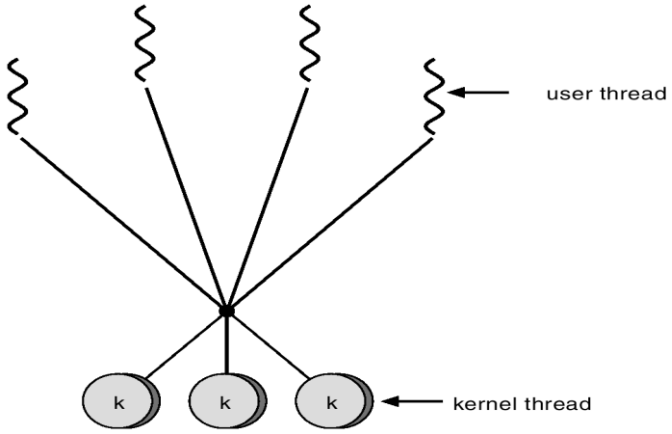


Gambar 2.4 *One by one* [9]

1) Model ini memetakan setiap *thread* tingkatan pengguna kesetiap *thread*.
2) Menyediakan lebih banyak *concurrency* dibanding model many to one. Keuntungannya sama dengan *threadkernel*.
3) Kelemahan model ini adalah setiap pembuatan *thread* pengguna memerlukan tambahan *threadkernel*. Karena itu, jika mengimplementasikan sistem ini maka akan menurunkan kinerja dari sebuah aplikasi sehingga biasanya jumlah *thread* dibatasi dalam sistem.
4) Contoh: windows NT/XP/2000, Linux, Solaris 9.

3. *Many to Many*

Berikut model multithreading *many to many* seperti yang tercantum pada gambar 2.5.



Gambar 2.5 *Many to Many* [9]

1) Model ini *memultipleks* banyak *thread* tingkatan pengguna ke *threadkernel* yang jumlahnya sedikit atau sama dengan tingkatan pengguna.
2) Model ini mengizinkan *developer* membuat *thread* sebanyak yang ia mau tetapi *concurrency* tidak dapat diperoleh karena hanya satu *thread* yang dapat dijadualkan oleh *kernel* pada satu waktu.
3) Keuntungan dari sistem ini ialah *kernelthread* yang bersangkutan dapat berjalan secara pararel pada *multiprosesor*.

2.2.2 Algoritma Penjadwalan

Penjadwalan CPU menyangkut penentuan proses – proses yang ada dalam *ready queue* yang akan dialokasikan pada CPU. Terdapat bebreapa algoritma penjadwalan yaitu:

2.2.2.1 First-Come First-Served (FCFS)

Algoritma ini merupakan algoritma penjadwalan yang paling sederhana yang digunakan CPU. Dengan menggunakan algoritma ini setiap proses yang berada pada status *ready* dimasukkan ke dalam FIFO *queue* sesuai dengan waktu kedatangannya. Proses yang tiba terlebih dahulu yang akan dieksekusi.

Algoritma ini juga menerapkan konsep non-*preemptive*, yaitu setiap proses yang sedang dieksekusi oleh CPU tidak dapat di *interrupt* oleh proses yang lain [9].

2.2.2.2 Shortest Job-First (SJF)

Algoritma ini mempunyai cara penjadwalanyang berbeda dengan FCFS.Dengan algoritma ini maka setiap proses yang ada di *readyqueue* akan dieksekusi berdasarkan *burst time* terkecil. Hal ini mengakibatkan *waiting time* yang pendek untuk setiap proses dan karena hal tersebut maka *waiting time* rata-ratanya juga menjadi pendek, sehingga dapat dikatakan bahwa algoritma ini adalah algoritma yang optimal.

Ada beberapa kekurangan dari algoritma ini yaitu:

1. Susahnya untuk memprediksi *burst time* proses yang akan dieksekusi selanjutnya.
2. Proses yang mempunyai *burst time* yang besar akan memiliki *waiting time* yang besar pula karena yang dieksekusi terlebih dahulu adalah proses dengan *burst time* yang lebih kecil.

Algoritma ini dapat dibagi menjadi 2 bagian yaitu:

a) *Preemptive*

Jika ada proses yang sedang dieksekusi oleh CPU dan terdapat proses di *readyqueue* dengan *burst time* yang lebih kecil daripada proses yang sedang dieksekusi tersebut, maka proses yang sedang dieksekusi oleh CPU akan digantikan oleh proses yang berada di *readyqueue* tersebut. *Preemptive* SJF sering disebut juga *Shortest-Remaining-Time-First Scheduling*.

b) *Non-preemptive*

CPU tidak memperbolehkan proses yang ada di *ready queue* untuk menggeser proses yang sedang dieksekusi oleh CPU meskipun proses yang baru tersebut mempunyai *burst time* yang lebih kecil [9].

2.2.2.3 *Priority Scheduling*

PriorityScheduling merupakan algoritma penjadwalan yang mendahulukan proses yang memiliki prioritas tertinggi. Setiap proses memiliki prioritasnya masing-masing. Prioritas suatu proses dapat ditentukan melalui beberapa karakteristik antara lain:

1. *Time limit*
2. *Memory requirement*
3. Akses file
4. Perbandingan antara I/O Burst dengan CPU Burst
5. Tingkat kepentingan proses

PriorityScheduling merupakan algoritma penjadwalan yang mendahulukan proses yang memiliki prioritas tertinggi. Setiap proses memiliki prioritasnya masing-masing.Prioritas suatu proses dapat ditentukan melalui beberapa karakteristik antara lain:

PriorityScheduling juga dapat dijalankan secara maupun non *preemptive*. Pada *preemptive*, jika ada suatu proses yang baru datang memiliki prioritas yang lebih tinggi daripada proses yang sedang dijalankan, maka proses yang sedang berjalan tersebut dihentikan, lalu CPU dialihkan untuk proses yang baru datang tersebut.Sementara itu, pada non-*preemptive*, proses yang baru datang tidak dapat mengganggu proses yang sedang berjalan, tetapi hanya diletakkan di depan *queue*. Kelemahan pada *PriorityScheduling* adalah dapat terjadinya *indefinite blocking* (*starvation*). Suatu proses dengan prioritas yang rendah memiliki kemungkinan untuk tidak dieksekusi jika terdapat proses lain yang memiliki prioritas lebih tinggi darinya.Solusi dari permasalahan ini adalah aging, yaitu meningkatkan prioritas dari setiap proses yang menunggu dalam *queue* secara bertahap [9].

2.2.2.4 *Round Robin*

Algoritma penjadwalan ini mirip dengan algoritma First Come FirstServed, tetapi proses ini memberi suatu batasan waktu untuk setiap proses yang disebut dengan *Timequantum*. *Timequantum* adalah suatu satuan waktu yang kecil. Jika proses yang sedang dieksekusi selesai dalam waktu kurang dari 1 *Timequantum*, tidak ada masalah. Tetapi jika proses berjalan melebihi 1 *Timequantum*, maka proses tersebut akan dihentikan,lalu digantikan oleh proses yang berikutnya. Proses yang dihentikan tersebut akan diletakkan di *queue* di urutan paling belakang.Permasalahan utama pada Round Robin adalah menentukan besarnya *Timequantum*. Jika *Timequantum* yang ditentukan terlalu

kecil, maka sebagian besar proses tidak akan selesai dalam 1 *Timequantum*.Hal ini tidak baik karena akan terjadi banyak *switch*, padahal CPU memerlukan waktu untuk beralih dari suatu proses ke proses lain (disebut dengan *contextswitch*es *time*). Sebaliknya, jika *Timequantum* terlalu besar, algoritma Round Robin akan berjalan seperti algoritma First Come First Served. *Timequantum* yang ideal adalah jika 80% dari total proses memiliki CPU *burst time* yang lebih kecil dari 1 *Timequantum* [9].

2.2.3 Jaringan

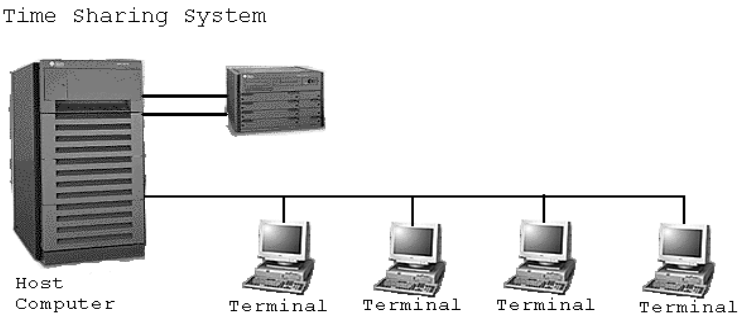
Jaringan komputer adalah sekelompok komputer otonom yang dihubungkan satu dengan yang lainnya dengan menggunakan protokol komunikasi melalui media transmisi atau media komunikasi sehingga dapat saling berbagi data informasi, program–program, *share* perangkat keras seperti *printer* atau *harddisk*.Tujuan dibangunnya suatu jaringan komputer adalah untuk membawa data informasi dari sisi pengirim menuju penerima secara cepat dan tepat tanpa adanya kesalahan melalui media transmisi atau media komunikasi tertentu [6].

2.2.3.1 Sejarah Jaringan

Konsep jaringan komputer lahir pada tahun 1940-an di Amerika dari sebuah proyek pengembangan komputer MODEL I di laboratorium Bell dan *group riset Harvard University* yang dipimpin profesor H. Aiken. Pada mulanya proyek tersebut ingin memanfaatkan sebuah perangkat komputer yang harus dipakai bersama. Untuk mengerjakan beberapa proses tanpa banyak membuang waktu kosong, maka dibuatlah proses beruntun (*Batch Processing*), sehingga beberapa program bisa dijalankan dalam sebuah komputer dengan dengan kaidah antrian.

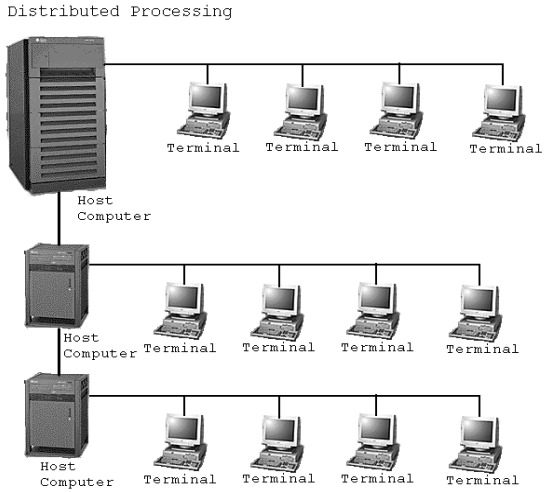
Ditahun1950-an ketika jenis komputer mulai membesar sampai terciptanya super komputer, maka sebuah komputer harus melayani beberapa terminal(Lihat Gambar 2.6). Untuk itu ditemukan konsep distribusi proses berdasarkan waktu yang dikenal dengan nama TSS (*Time Sharing System*), maka

untuk pertama kali bentuk jaringan (*network*) komputer diaplikasikan. Pada sistem TSS beberapa terminal terhubung secara seri ke sebuah *host* komputer. Dalam proses TSS mulai nampak perpaduan teknologi komputer dan teknologitelekomunikasi yang pada awalnya berkembang sendiri-sendiri [6].



Gambar 2.6 Jaringan Komputer Model TSS

Memasuki tahun 1970-an, setelah beban pekerjaan bertambah banyak dan harga perangkat komputer besar mulai terasa mahal, maka mulailah digunakan konsep proses distribusi (*distributed processing*). Seperti pada Gambar 2.6 dalam proses ini beberapa *host* komputer mengerjakan sebuah pekerjaan besar secara paralel untuk melayani beberapa terminal yang tersambung secara seri disetiap *host* komputer. Dalam proses distribusi sudah mutlak diperlukan perpaduan yang mendalam antara teknologi komputer dan telekomunikasi, karena selain proses yang harus didistribusikan, semua host komputer wajib melayani terminal-terminalnya dalam satu perintah dari komputer pusat.



Gambar 2.7 Jaringan Komputer Model Distributed Processing

Ketika harga-harga komputer kecil sudah mulai menurun dan konsep proses distribusi sudah matang, maka penggunaan komputer dan jaringannya sudah mulai beragam dari mulai menangani proses bersama maupun komunikasi antar komputer (*peer to peer system*) saja tanpa melalui komputer pusat. Untuk itu mulailah berkembang teknologi jaringan lokal yang dikenal dengan sebutan LAN (*Local Area Network*). Demikian pula ketika Internet mulai diperkenalkan, maka sebagian besar LAN yang berdiri sendiri mulai berhubungan dan terbentuklah jaringan raksasa WAN (*Wide Area Network*) [6].

2.2.3.2 Jenis – Jenis Jaringan Komputer

Jaringan komputer yang digunakan dewasa ini dapat dikelompokkan berdasarkan luasnya area yang dapat dijangkau. Luasnya area ini pada mulanya ditentukan berdasarkan jarak jangkau dalam satuan meter.Namun dalam perkembangan selanjutnya berdasarkan peralatan dan fasilitas. Penggunaan peralatan–peralatan yang dimaksud adalah peralatan–peralatan *inter* jaringan seperti *Repeater, Router* atau *Gateway* dari peralatan tersebut dapat dilihat jangkauan area jaringan dan luasan segmen jaringan yang dibangun. Berdasarkan kriteria–kriteria tersebut secara umum jaringan komputer dapat digolongkan sebagai berikut [6]:

a. *Local Area Network* (LAN)

   Sebuah LAN adalah jaringan yang dibatasi oleh area yang relatif kecil, umumnya dibatasi oleh area lingkungan seperti sebuah perkantoran di sebuah gedung, atau sebuah sekolah, dan biasanya sekitar 1 km persegi. Komputer-komputer yang terhubung ke dalam jaringan itu biasanya disebut dengan *workstation.* Kebanyakan LAN menggunakan media kabel untuk menghubungkan antara satu komputer dengan komputer lainnya.

b. *Metropolitan Area Network* (MAN)

   Sebuah MAN, biasanya meliputi area yang lebih besar dari LAN, misalnya antar wilayah dalam satu provinsi. Dalam hal ini jaringan menghubungkan beberapa buah jaringan-jaringan kecil ke dalam lingkungan area yang lebih besar, sebagai contoh yaitu : jaringan Bank dimana beberapa kantor cabang

sebuah Bank di dalam sebuah kota besar dihubungkan antara satu dengan lainnya.

c. *Wide Area Network* (WAN)

Wide Area Networks(WAN) adalah jaringan yang lingkupnya biasanya sudah menggunakan sarana satelit ataupun kabel bawah laut sebagai contoh keseluruhan jaringan suatu Bank yang ada di Indonesia ataupun yang ada di negara - negara lain. menggunakan sarana WAN, Sebuah Bank yang ada di Bandung bisa menghubungi kantor cabangnya yang ada di Jepang, hanya dalam beberapa menit. Biasanya WAN agak rumit dan sangat kompleks, menggunakan banyak sarana untuk menghubungkan antara LAN dan WAN ke dalam komunikasi global seperti Internet. Tapi bagaimanapun juga antara LAN, MAN dan WAN tidak banyak berbeda dalam beberapa hal, hanya lingkup areanya saja yang berbeda satu diantara yang lainnya.

d. Internet

Internet merupakan jaringan Global Internasional di seluruh dunia. Biasanya teknologi ini menggunakan suatu fasilitas layanan yang biasa kita sebut *World Wide Web* (WWW).

2.2.3.3 Tipe – Tipe Jaringan Komputer

Terdapat dua macam model jaringan komputer, yaitu :

a. Model *Peer to Peer*

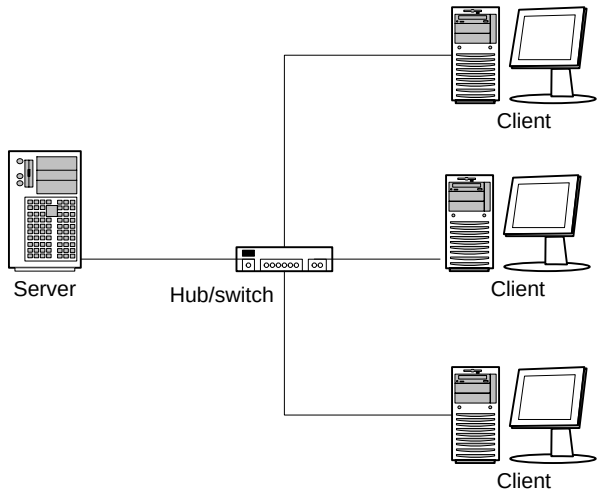
Pada model jaringan terdapat hanya dua komputer yang saling terhubung. Masing-masing komputer mempunyai kedudukan yang setara atau setingkat sehingga komputer manapun dapat memulai (*initiate*) dan mengendalikan hubungan dalam jaringan. Model Jaringan ini merupakan jenis yang paling sederhana. Model jaringan peer to peer dapat terlihat pada gambar 2.8 [6].



Gambar 2.8 Model Jaringan *Peer to Peer*

b. Model *ClientServer*

Konsep Jaringan *client-server* ini membedakan dengan jelas kedudukan suatu komputer, mana yang dapat memberikan layanan jaringan (*server*) dan mana yang hanya meminta layanan (*client*). Jika suatu komputer di*install* sebagai *server* yang memberikan segala sumber daya (*resource*), contohnya : *printer*, *modem* kepada komputer lain (*client)* yang terhubung ke jaringan. Untuk dapat saling berkomunikasi antara *server* dan *client*, maka digunakan suatu aplikasi jaringan yang dinamakan *client-server program,* dimana *server* menggunakan aplikasi jaringan yang disebut *server program*, sementara *client* menggunakan *client program* untuk dapat berkomunikasi dengan *server program* pada *server*. Model jaringan *client-server* dapat terlihat pada gambar 2.9 [6].



Gambar 2.9 Model Jaringan *Client-Server*

Beberapa sifat dari *client-server* diantaranya sebagai berikut :

1. *Server* dan *Client* berada pada posisi serta proses yang berbeda.
2. *Server* dan *Client* dapat dijalankan pada mesin yang sama atau berbeda.
3. Setiap proses dapat menyembunyikan informasi tersendiri
4. Komunikasi yang dilakukan bisa bersifat protokol *peer to peer*.

Koneksitas *client ke Server :*

1. *Server* harus terlebih dahulu dijalankan (*running*).
2. *Server* harus menentukan *port* untuk koneksi yang masuk.

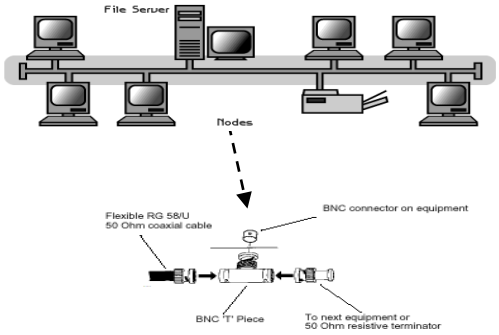
3. *Client* harus mengetahui nama *host* dari *server* beserta *port* yang digunakan untuk koneksi.
4. *Client* mengirim pesan ke *server* untuk melakukan suatu pekerjaan (*task*). *Server* dan *Client* bersifat *Asimetrik.*

2.2.3.4 Topologi Jaringan

Topologi jaringan merupakan suatu struktur atau bagaimana menghubungkan antara simpul dan pusat dalam suatu jaringan. Tiap struktur mempunyai keuntungan dan kerugiannya masing-masing. Topologi yang umumnya sebagai berikut :

a. Topologi Linear BUS - Garis Lurus

Topologi sebuah *LinearBUS* / garis lurus terdiri dari satu jalur kabel utama (*backbone*) dimana pada masing-masing ujungnya diberikan sebuah *terminator*. Semua *nodes* pada jaringan (*file server, workstation,* dan perangkat lainnya) terkoneksi ke sebuah kabel utama. Jaringan-jaringan *Ethernet* dan *Localtalk* menggunakan topologi *linear* ini. Topologi *Linear BUS /* garis lurus dapat dilihat pada Gambar 2.10.

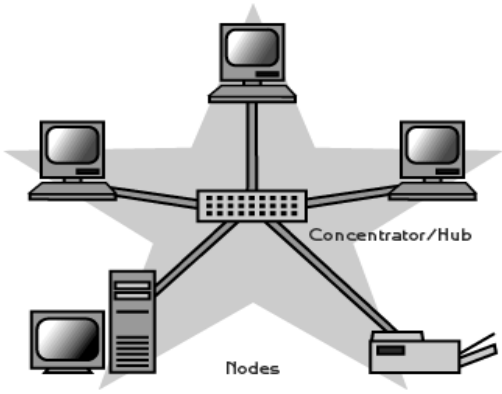


Gambar 2.10 Topologi Linear BUS / Garis Lurus

b. Topologi Star – Bintang

Topologi model ini didesain dimana setap *node*(*file server, workstation*, dan perangkat lainnya) terkoneksi ke jaringan melewati sebuah *concentrator* baik berupa HUB ataupun *SWITCH*. Data yang terkirim ke jaringan akan melewati *concentrator* sebelum melanjutkan ke tempat tujuannya. *Concentrator* akan mengatur dan mengontrol keseluruhan fungsi jaringan.

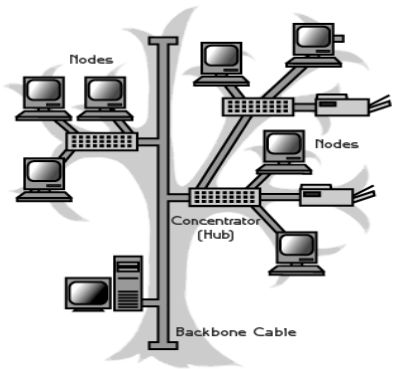
Dia juga bertindak sebagai *repeater* dalam skala kecil dari aliran data.Konfigurasi pada jaringan model ini menggunakan kabel *TwistedPair*. Topologi jaringan Bintang / *Star* dapat dilihat pada Gambar 2.11.



Gambar 2.11 Topologi Jaringan Bintang/Star

c. Topologi Tree – Pohon

Topologi model ini merupakan perpaduan antara topologi *LinearBUS* / Garis Lurus dan Bintang, yang terdiri dari kelompok-kelompok dari *workstation* konfigurasi Bintang yang terkoneksi kekabel utama yang menggunakan topologi *LinearBUS* / Garis Lurus.Topologi ini memungkinkan untuk pengembangan jaringan yang telah ada, dan memungkinkan sebuah perusahaan mengkonfigurasi jaringan sesuai dengan kebutuhannya. Topologi jaringan pohon dapat dilihat pada Gambar 2.12.

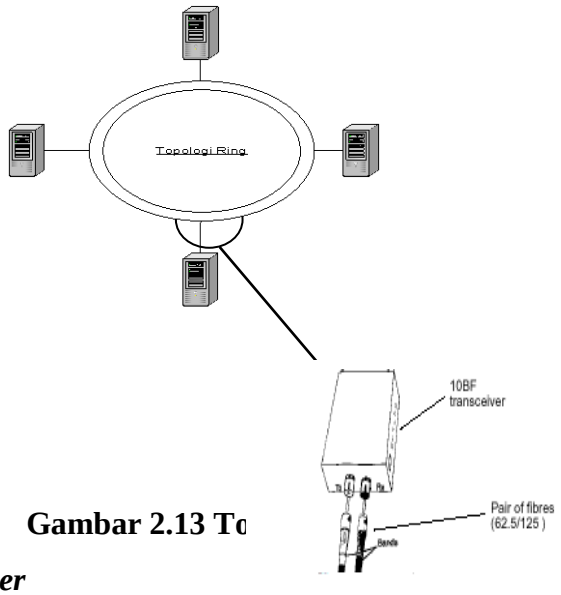


Gambar 2.12 Topologi Jaringan Pohon

d. Topologi Ring – Cincin

Topologi *Ring* semua *workstation* dan *server* dihubungkan sehingga terbentuk suatu pola lingkaran atau cincin. Tiap *workstation* ataupun *server*akan

menerima dan melewatkan informasi dari satu komputer ke komputer lain, bila alamat- alamat yang dimaksud sesuai maka informasi diterima dan bila tidak informasi akan dilewatkan. Topologi jaringan ring dapat dilihat pada Gambar 2.13.



Gambar 2.13 To

2.2.4 *Client-Server*

Model *clientserver* sudah mendominasi untuk penggunaan sistem terdistribusi sejak tahun 1980.Perkembangan model ini pada akhirnya memungkinkan untuk *user* bekerja pada computer masing – masing yang terhubung dengan jaringan lokal (LAN). Inti dari model ini adalah bagaimana cara mengakses, menggunakan dan membagi (*share*) suatu sumber daya (*resource*) yang terdapat pada komputer lain. Pada tahun 1980, komputer dikendalikan oleh sebuah sistem operasi yang memiliki *kernelmonolithic* , dimana semua *servis* merupakan bagian dari keseluruhan perangkat lunak. Pada tahun 1990 model *client – server* sudah dapat digunakan untuk mengembangkan banyak aplikasi. Hal ini dikarenakan model tersebut akan mengurangi biaya baik dalam pengimplementasian dan biaya perawatan [6].

2.2.4.1 Konsep Dasar *Client–server*

Pada konsep ini terdapat dua buah proses, *client* yang meminta (*request*) sebuah *servis* dari proses yang lain, dan *server* yang merupakan penyedia *servis*. *Server* akan melakukan kegiatan yang diminta dan mengirim umpan balik sebagai

respons. *Respons* ini dapat berupa hasil, konfirmasi, maupun peringatan tentang kegagalan dari suatu operasi.

Sebuah model *client server* yang lebih terperinci memiliki 3 komponen :

1. *Service* : merupakan sebuah *entitas* perangkat lunak yang berjalan pada satu atau lebih mesin. *Service* menyediakan suatu *abstraksi* dari rentetan operasi yang diminta oleh perangkat lunak.
2. *Server* : sekumpulan *Service* yang berjalan pada sebuah mesin.
3. *Client* : sebuah *entitas* perangkat lunak yang mengeksploitasi *Service* yang disediakan oleh *server*. Sebuah *client* dapat berinteraksi dengan *user* secara langsung, tetapi tidak harus.

Ada 3 masalah utama yang terdapat pada model *client – server* :

1. Jika komputer yang menjadi *server* mengalami gangguan sehingga tidak dapat beroperasi sebagaimana mestinya, maka *servis* yang diberikan oleh komputer tersebut pun tidak dapat dijalankan. Oleh karena itu, *reliability* dan *availability* dari suatu operasi yang bergantung pada banyak *server* merupakan suatu produk yang bergantung pada perangkat keras dan jaringan komunikasi.
2. Permasalahan kedua yaitu adanya kemungkinan timbul potensi *bottleneck*. Hal ini terjadi karena penambahan *client* pada sistem.
3. Permasalahan terakhir timbul pada saat mengimplementasikan banyak fungsi yang serupa pada suatu sistem untuk menjaga performansi dan konsistensi. Hal ini meningkatkan biaya keseluruhan dari sistem distribusi tersebut.

2.2.5 Konsep Perancangan Berorientasi Objek

Teknologi objek menganalogikan sistem aplikasi seperti kehidupan nyata yang didominasi oleh objek. Didalam membangun sistem berorientasi objek akan menjadi lebih baik apabila langkah awalnya didahului dengan proses analisis dan perancangan yang berorientasi objek. Tujuannya adalah mempermudah *programmer* didalam mendesain program dalam bentuk objek-objek dan hubungan antar objek tersebut untuk kemudian dimodelkan dalam sistem nyata.

Suatu perusahaan *software* yaitu *Rational Software*, telah membentuk *konsarium* dengan berbagai organisasi untuk meresmikan pemakaian *Unifed*

*Modelling Language (*UML*)* sebagai bahasa standar dalam *Object Oriented Analysist Design (OOAD)* [7].

2.2.5.1 Konsep UML *(Unified Modeling Language)*

UML (*Unified Modeling Language*) merupakan pengganti dari metode analisis berorientasi objek dan desain berorientasi objek (OOA&D) yang dimunculkan sekitar akhir tahun 80-an dan awal tahun 90-an. UML merupakan gabungan dari metode Booch, Rumbaugh (OMT) dan Jacobson. Tetapi UML ini akan mencakup lebih luas daripada OOA&D. Pada pertengahan pengembangan UML dilakukan standarisasi proses dengan OMG (*Object Management Group*) dengan harapan UML akan menjadi bahasa standar pemodelan pada masa yang akan datang.

UML disebut sebagai bahasa pemodelan bukan metode. Kebanyakan metode terdiri paling sedikit prinsip, bahasa pemodelan dan proses. Bahasa pemodelan (sebagian besar grafik) merupakan notasi dari metode yang digunakan untuk mendesain secara cepat.

Bahasa pemodelan merupakan bagian terpenting dari metode. Ini merupakan bagian kunci tertentu untuk komunikasi. Jika anda ingin berdiskusi tentang desain dengan seseorang, maka Anda hanya membutuhkan bahasa pemodelan bukan proses yang digunakan untuk mendapatkan desain. UML merupakan bahasa standar untuk penulisan *blueprint software* yang digunakan untuk visualisasi, spesifikasi, pembentukan dan pendokumentasian alat-alat dari sistem perangkat lunak [7].

2.2.5.2 Sejarah UML

UML dimulai secara resmi pada oktober 1994, ketika Rumbaugh bergabung dengan Booch pada Relational Software Corporation. Proyek ini memfokuskan pada penyatuan metode Booch dan OMT. UML versi 0.8 merupakan metode penyatuan yang dirilis pada bulan Oktober 1995. Dalam waktu yang sama, Jacobson bergabung dengan Relational dan cakupan dari UML semakin luas sampai diluar perusahaan OOSE. Dokumentasi UML versi 0.9

akhirnya dirilis pada bulan Juni 1996. Meskipun pada tahun 1996 ini melihat dan menerima *feedback* dari komunitas *Software Engineering*.

Dalam waktu tersebut, menjadi lebih jelas bahwa beberapa organisasi perangkat lunak melihat UML sebagai strategi dari bisnisnya, kemudian dibangunlah UML Consortium dengan beberapa organisasi yang akan menyumbangkan sumber dayanya untuk bekerja, mengembangkan, dan melengkapi UML. Di sini beberapa *partner* yang berkontribusi pada UML 1.0, diantaranya Digital Equipment Corporation, Hewlett-Packard, I-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Relational, Texas Instruments dan Unisys. Dari kolaborasi ini dihasilkan UML 1.0 yang merupakan bahasa pemodelan yang ditetapkan secara baik, *expressive*, kuat, dan cocok untuk lingkungan masalah yang luas. UML 1.0 ditawarkan menjadi standarisasi dari Object Management Group (OMG) dan pada Januari 1997 dijadikan sebagai standar bahasa pemodelan.

Antara Januari–Juli 1997 gabungan group tersebut memperluas kontribusinya sebagai hasil respon dari OMG dengan memasukkan Adersen Consulting, Ericsson, Object Time Limeted, Platinum Technology, Ptech, Reich Technologies, Softeam, Sterling Software dan Taskon. Revisi dari versi UML (versi 1.1) ditawarkan kepada OMG sebagai standarisasi pada bulan Juli 1997. Dan pada bulan September 1997,versi ini dierima oleh OMG *Analysis* dan *Design Task Force* (ADTF) dan OMG *Architecture Board* dan Akhirnya pada Juli 1997 UML versi 1.1 menjadi standarisasi.

Pemeliharaan UML terus dipegang oleh OMG *Revision Task Force* (RTF) yang dipimpin oleh Cris Kobryn. RTP merilis editorial dari UML 1.2 pada Juni 1998 dan pada tahun 1998 RTF juga merilis UML 1.3 disertai dengan *user guide* dan memberikan *technical cleanup* [7].

2.2.5.3 Tipe-Tipe Diagram UML

UML memiliki tipe-tipe diagram, diantaranya yaitu sebagai berikut :

a. *Use Case Diagram*

Use Case Diagram digunakan untuk menggambarkan konteks sistem yang akan dibangun dan fungsionalitas yang disediakan oleh sistem itu. Use Case Diagram juga menggambarkan siapa (atau apa) berinteraksi dengan sistem. Di dalam sistem hanya terdapat satu *user*.

b. *Class Diagram*

Class diagram digunakan untuk menunjukan hubungan dan eksistensi sebuah class dari sudut pandang logis dalam sistem. Sebuah Class Diagram merepresentasikan struktur class yang membentuk arsitektur dan perilaku sistem dalam proses desain dan analisis. *Class diagram* memberikan gambaran secara menyeluruh tentang object dan class serta relasinya pada arsitektur sistem yang dibangun. *Class diagram* menyediakan fasilitas berbagi model sehingga model yang telah dibangun bisa digunakan kembali (reusable) untuk pemodelan yang lebih rinci pada perilaku dinamis. Notasi pada *class diagram* bisa digunakan untuk menjelaskan *Object Oriented Programming*, sehingga semua konsep *Object Oriented Programming* bisa di jelaskan menggunakan notasi *Class diagram.*

c. *Statechart Diagram*

Statechart Diagram menggambarkan semua *state* yang dimiliki oleh suatu objek dari suatu class dan keadaan yang menyebabkan state berubah.

d. *Sequence Diagram*

Sequence diagram merupakan suatu diagram yang menggambarkan interaksi antar objek dan mengindikasikan komunikasi diantara objek-objek tersebut. Diagram ini juga menunjukkan serangkaian pesan yang dipertukarkan oleh objek-objek yang melakukan suatu tugas atau aksi tertentu. Objek-objek tersebut kemudian diurutkan dari kiri ke kanan, aktor yang menginisialisasi interaksi biasanya disimpan di paling kiri dari diagram.

e. *Collaboration Diagram*

Collaboration Diagram menggambarkan kolaborasi dinamis seperti *sequence diagram*. Dalam menunjukan pertukaran pesan, collaboration diagram menggambarkan objek dan hubungannya.

f. *Activity Diagram*

Activity diagram merupakan cara menggambarkan proses bisnis dan urutan aktivitas dalam sebuah proses yang dipakai pada *business modeling* untuk memperlihatkan urutan aktifitas proses bisnis. Struktur diagram ini mirip *flowchart* atau *Data Flow Diagram* (DFD) pada perancangan terstruktur. *Activity diagram* sangat bermanfaat apabila kita membuat diagram ini terlebih dahulu dalam memodelkan sebuah proses untuk membantu memahami proses secara keseluruhan.

g. *Component Diagram*

Menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan diantaranya. Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lainnya.

h. *Package Diagram*

Package diagram digunakan untuk menjelaskan hubungan modularitas, enkapsulasi, dan abstraksi pada pengembangan sistem yang kompleks. *Package diagram* juga digunakan untuk menjelaskan hubungan *dependency* dan *visibility* antar *class* yang memiliki asosiai secara modular dengan *class* lain dalam sistem. *Package diagram* memungkinkan perancangan sistem bisa di mengerti dengan tingkat pemahaman yang lebih tinggi pada kolaborasi antar elemen model. Dengan menggunakan *package diagram* perancangan *class diagram* bisa di modularisasi secara lebih terstruktur. *Package diagram* sangat dibutuhkan ketika *class diagram* yang dibangun sudah semakin kompleks, sehingga penggambaran *class diagram* bisa dibangun secara terpisah dengan menggunakan *Package diagram* untuk mewakili hubungan yang antar class pada *package* yang berbeda.

i. *Deployment Diagram*

Deployment diagram digunakan untuk menunjukan alokasi artefak pada node dalam desain fisik sebuah sistem. Sebuah *deployment diagram* mewakili sebuah gambaran ke dalam struktur artefak suatu sistem. *Deployment diagram* lebih berfokus pada aspek fisik dari *object-oriented system*. Dengan menggunakan *deployment diagram*, gambaran mengenai distribusi komponen dan relasinya dengan sistem maupun *platform* tempat sistem berjalan bisa digambarkan dengan lebih jelas, sehingga dengan menggunakan *deployment diagram*, gambaran akan interaksi antara *software* dan *hardware* pada sistem akan terlihat lebih jelas [7].

2.2.6 Jaringan Semantik

Jaringan semantik adalah gambaran pengetahuan grafis yang menunjukkan hubungan antar berbagai objek, terdiri dari lingkaran-lingkaran yang dihubungkan dengan anak panah yang menunjukkan objek dan informasi tentang objek-objek tersebut. Gambar berikut menunjukkan representasi pengetahuan menggunakan jaringan semantik.

2.2.7 MySQL

MySQL adalah sebuah sistem manajemen database relasi *(relational database management system)* yang bersifat "terbuka" *(open source).*Terbuka maksudnya adalah MySQL boleh di-*download* oleh siapa saja, baik versi kode program aslinya maupun versi binernya dan dapat digunakan secara (relatif) gratis baik untuk dimodifikasi sesuai dengan kebutuhan seseorang maupun sebagai suatu program aplikasi komputer.

2.2.7.1 Sejarah MySQL

Sejarah MySQL yang merupakan hasil buah pikiran dari Michael "Monty" Widenius, David Axmark, dan Alan Larson dimulai tahun 1995.Mereka bertiga kemudian mendirikan perusahaan bernama MySQL AB di Swedia. Tujuan awal

ditulisnya program MySQL adalah untuk mengembangkan aplikasi web yang digunakan salah satu klien MySQL AB.

2.2.8 Borland Delphi

Delphi adalah paket bahasa pemrograman yang bekerja dalam sistem operasi Windows.Delphi merupakan bahasa pemrograman yang mempunyai cakupan kemampuan yang luas dan sangat canggih.

Kelebihan-kelebihan yang dapat diambil ketika seorang pengembang perangkat lunak menggunakan Borland Delphi adalah :

1. Delphi mendukung Pemrograman Berorientasi Objek (*Object Oriented Programming*).
2. Hasil dari proses kompilasi berupa sebuah file yang dapat dieksekusi (*executabel file*) sehingga mempermudah dalam pendistribusian program dan mengurangi banyaknya *file* pendukung.
3. Delphi menyediakan banyak sekali komponen yang dapat digunakan. Selain itu banyak juga komponen yang bersumber dari pihak ketiga yang biasanya disertai dengan dokumentasi, *source code* dan lain-lain.Komponen dari pihak ketiga bisa yang komersil atau *free*.
4. Mendukung banyak database *server* (*MySQL, SQL Server*, *Interbase*, *Oracle* dll) sehingga dapat mempermudah dalam membuat aplikasi database.
5. Borland Delphi menyediakan fasilitas yang luas mulai dari fungsi membuat form hingga untuk menggunakan format *file* berbasis data yang popular seperti *Dbase* dan *Paradoks*.

2.2.8.1 Keis*time*waan MySQL

Sebagai *database server* yang memiliki konsep *database modern*, MySQL memiliki banyak keis*time*waan. Berikut ini beberapa keis*time*waan yang dimiliki oleh MySQL.

1. Portability

MySQL dapat berjalan stabil pada berbagai sistem operasi seperti Windows, Linux, FreeBSD, Mac OS X *Server*, dan lain-lain.

2. *Open Source*

MySQL didistribusikan secara *Open Source*, sehingga dapat digunakan secara bebas.

3. *Multi-user*

MySQL dapat digunakan oleh beberapa *User*dalam waktu yang bersamaan tanpa mengalami masalah.Hal ini memungkinkan sebuah *database server* MySQL dapat diakses *client* secara bersamaan.

4. *Performance Tuning*

MySQL memiliki kecepatan yang tinggi dalam menangani *query*, dengan kata lain dapat memproses lebih banyak SQL per satuan waktu.

5. *Column Types*

MySQL memiliki tipe kolom yang sangat kompleks, seperti *signed/unsigned integer, float, double, char, varchar, text, blob, date, time, timestamp, year, set* dan *enum.*

6. *Command* dan *Function*

MySQL memiliki operator dan fungsi secara penuh yang mendukung perintah SELECT dan WHERE dalam *query.*

7. *Security*

MySQL memiliki lapisan lapisan sekuritas seperti *level subnetmask,*nama*host,* dan izin akses *User*dengan *sistem* perizinan yang mendetail serta *password* terenkripsi.

8. *Scalability* dan Limits

MySQL mampu menangani *database* dengan skala besar, dengan jumlah records lebih dari 50 juta dan 60 juta tabel serta 5 miliar baris. Selain itu, batas indeks yang dapat ditampung mencapai 32 indeks pada setiap tabelnya.

9. *Connectivity*

MySQL dapat melakukan koneksi dengan *client* menggunakan *protocol* TCP/IP, Unix Socet (UNIX), atau Named Pipes (NT).

10. *Localisation*

Deteksi pesan kesalahan pada *client* dengan menggunakan lebih dari 20 bahasa

11. *Interface*

Interface terhadap berbagai aplikasi dan bahasa pemrograman dengan menggunakan fungsi API (*Application Programming Interface*)

12. *Client* dan *Tools*

Dilengkapi dengan berbagai *Tool* yang dapat digunakan untuk administrasi *database*, dan pada setiap *tool* yang ada disertakan petunjuk *online*.

13. Struktur Tabel

Struktur tabel yang lebih fleksibel dalam menangani ALTER TABEL, dibanding database lainnya semacam PostgreSQL ataupun Oracle.